Scalable Big Data analytics in the cloud: Challenges and Solutions.

**Speaker:** Abdul Quamar*

**Joint work with:**
Amol Deshpande*, Jimmy Lin*, Badrish Chandramouli†, Jonathan Goldstein†

* University of Maryland, College Park
† Microsoft Research, Redmond
Introduction

• **Big data in the cloud**
  • Large volumes of data being stored and processed in the cloud
  • Diversity: Structured Vs unstructured, Sensor data streams, Multimedia (audio, video) Click streams, Log files, etc
  • Analytics: derive value out of the massive data

• **Graph analytics**
  • Information networks have become ubiquitous
  • Network data is most naturally represented as a graph
  • Supporting complex analytics over large volumes of graph-structured data remains a challenge

• **Analytics in the relational domain**
  • Most of the Big data is still relational or set-oriented
  • Complex analytics over large volumes in the cloud expensive
  • Reducing the cost of analytics is a primary challenge
Our Solutions

• **NScale: Processing Big Graph-structured Data in the Cloud**
  • Increasing interest in complex graph analytics on very large graphs
  • Global:
    • Shortest paths, connected components, PageRank
  • Local (or neighborhood-centric):
    • Typically interested in properties/behavior of individual or a group of nodes
    • Analysis of social circles, link prediction, Ego-centric pattern census, motif counting, local clustering coefficients, etc
  • **NScale**: A scalable graph processing framework that supports the spectrum of complex graph analysis tasks

• **NOW! : Progressive Big Data Analytics in the Cloud**
  • Progressive analytics:
    • Provides early results using significantly fewer resources
    • Allows users to get to and potentially end (and possibly refine) computations early
  • **Now!**: Provides a scalable progressive solution to reduce the cost of analytics on large scale data
Outline

• Graph analytics

• Big data analytics in the relational domain

• Conclusion
Graph Analytics: Current Programming Frameworks

- **MapReduce-based**
  - Gbase, Pegasus, Hadapt
  - Use Map-Reduce as the underlying distributed processing framework
- **Disadvantages:**
  - Not intuitive to program graph analysis tasks using MapReduce
  - Each "traversal" effectively requires a new MapReduce phase: Inefficient
Graph Analytics: Current Programming Frameworks

• Vertex-centric iterative programming frameworks
  • Application programs: Vertex-level kernel computation written by the user
  • Sharing of state:
    • Message passing /shared memory abstractions
• Program Execution:
  • Synchronous: Pregel, Giraph (BSP)
  • Asynchronous: GraphLab, GRACE
  • Hybrid: Kineograph
• Issues:
  • Not sufficient or natural for many query analysis tasks (Ego network analysis)
  • No inherent support for applications that require analytics on the neighborhoods of a subset of nodes
  • E.g. NHs would need to be re-constructued at each node through message passing
  • Inefficient for analytics that require traversing beyond 1-hop neighbors
NScale: An End-to-end Distributed Graph Programming Framework

- **Users/application programs specify:**
  - Neighborhoods or subgraphs of interest
  - A kernel computation to operate upon those subgraphs

- **Framework:**
  - Extracts the relevant subgraphs from underlying data and loads in memory
  - Execution engine: Executes user computation on materialized graphs subgraphs
  - Communication: Shared state/message passing
NScale: System Architecture

Users

Analysts

Applications/Visualization Tools

NScale User API

Underlying graph data

Flat files

<K1,V1>

<K2,V2>

...

Key-Value stores

Graph Extraction and Loading

MapReduce (Apache Yarn)

Graph extraction

Graph analytics

In-Memory Distributed Execution Engine

Output

Output Materialization Checkpointing

Special purpose indexes

Analysts

Applications/Visualization Tools

In-Memory Distributed Execution Engine

Output Materialization Checkpointing

Special purpose indexes
Example: Local Clustering coefficient (LCC)

Nsake User API (Datalog, BluePrints): Query: Compute LCC for nodes where node.color=red

Underlying graph data on HDFS → Graph Extraction and Loading → Subgraphs in Distributed Memory → Graph analytics → Output

MapReduce (Apache Yarn) → Graph extraction

Distributed Execution Engine

Output Materialization

Checkpointing

Mechanics/Challenges

Consistency Execution Models:
- Synchronous
- Asynchronous

Sharing of State:
- Shared memory/messaging passing

Exploiting Overlap

Graph cut and set bin packing

Supernodes
NScale: Summary

- User writes programs at the abstraction of a graph
  - More intuitive for graph analytics

- Scalability: Only relevant portions of the graph data loaded into memory
  - User can specify subgraphs of interest, and select nodes or edges based on properties
  - E.g. Edges with recent communication

- Generalization: Flexibility in subgraph definition
  - Handle vertex-centric programs
    - Subgraph: vertex and associated edges
  - Global programs
    - Subgraph is the entire graph

- Applicability:
  - Captures mechanics of a common graph analysis/cleaning tasks
  - Complex analytics:
    - Union or intersection of neighborhoods (Link prediction, Entity resolution)
    - Induced subgraph of a hashtag (Influence analysis on hashtag ego networks)
Outline

• Graph analytics

• Big data analytics in the relational domain

• Conclusion
Big Data Analytics in the Relational Domain

• Existing systems for big data analytics
  • Map-Reduce (and all its flavors: Haloop, MOL etc), Hive
  • Parallel DBMS
  • Shark, Spark, etc.

• Users run analytics jobs on large expensive clusters
  • Analytics jobs take long times to complete
  • Expensive due to the pay-as-you-go nature of resources (e.g., in Cloud)
  • Exacerbated by need for interactive exploratory query experience

• Bottom line: Big data analytics allows one user to burn significant amounts of money very easily
Reducing the Cost of Analytics

• **Do less work**
  - Produce results over partial data
  - Refine as more data is read
  - User can stop computations early

• **Approximate query processing**
  - Tries to automate the production of approximate results
  - Not feasible for arbitrary queries

• **How do data scientists work today?**
  - They understand the nature of their data
  - Create & work with sequence of progressive data samples (increasing size)
  - They control their sampling strategies; chosen based on data & queries
A Progressive Sampling Model

- Define a logical progress domain \([0, \infty)\)
- A tuple has a progress-start value \((P^{\uparrow+})\)
  - Logical point where a tuple enters the computation
- Tuples may also leave the computation
  - Example: Average operator has to remove old & report new average after processing more tuples
  - Represented as a progress-end value \((P^{\uparrow-})\)
- Thus, every tuple has a progress-interval \([P^{\uparrow+}, P^{\uparrow-}]\)
Example: computing click-through-rate

- Observation:
  - Our model precisely matches temporal semantics used in streaming engines today
  - We use StreamInsight to implement progressive
E.g. Implementing Stratified Sampling

AlterLifetime
\( vs = \lceil \frac{seq}{k} \rceil \)

TemporalUnion

<table>
<thead>
<tr>
<th>Keys</th>
<th>#Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>1000</td>
</tr>
<tr>
<td>K2</td>
<td>500</td>
</tr>
<tr>
<td>K3</td>
<td>800</td>
</tr>
<tr>
<td>K4</td>
<td>20</td>
</tr>
</tbody>
</table>

Group & Apply

<table>
<thead>
<tr>
<th>Value</th>
<th>Ps</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>0</td>
</tr>
<tr>
<td>V2</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>V10</td>
<td>0</td>
</tr>
<tr>
<td>V11</td>
<td>1</td>
</tr>
<tr>
<td>V12</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>V20</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keys</th>
<th>#Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>10</td>
</tr>
<tr>
<td>K2</td>
<td>10</td>
</tr>
<tr>
<td>K3</td>
<td>10</td>
</tr>
<tr>
<td>K4</td>
<td>10</td>
</tr>
</tbody>
</table>
Scaling Out on a Cluster

• Vanilla map-reduce does not fit
  • Not pipelined
  • Does not understand/propagate progress

• Prototyped **Now**!
  • A progress-aware map-reduce framework
  • Works on Windows Azure
  • Explicit progress as a first-class citizen
  • Supports “progress-aware reducers” (may run temporal streaming engine inside)

• Leverages related work
  • Distributed streams, fault tolerance, RDDs, …
Now! Architecture and Design

Pipelined progress-aware data flow

Progressive Sampling
• Input data partitioned into input splits.
• Data tuples assigned progress intervals.

Progress-aware batching
• Map phase produces progress batches partitioned by key.

Progressive Data Shuffle
• Data shuffled between the mappers and reducers in terms of progress-batches without sorting.
• Downstream reducers consume the progress batches.
Now! Architecture and Design

Progress Aware Merge
- Ensures flow of data in strict progress order along all paths in the framework.

Progress Aware Reducer
- StreamInsight as a progressive reducer to handle progressive relational queries.
Results

Scalability with increase in data size

Throughput Scalability

Throughput scale-up factor

Scale-up scale-up factor

Data size (GB)

# Machines

20 (1X)

30 (1.5X)

45 (2.25X)

60 (3X)

74 (3.7X)
Results

![Top-k Convergence graph]

- **Precision**
- **Progress**
- **Top-k Convergence**
- **Convergence**
  - k-1000
  - k-500
  - k-100
  - k-50
  - k-10
Now!: Summary

• Gives control to data scientists
  • Choose from a variety of domain-specific sampling strategies
  • Example: load dimension tables before fact tables for meaningful early results
  • Compute confidence intervals as part of query

• System provides meaningful composable semantics

• Determinism
  • Progressive results are function of input data and query alone
  • Gives to progressive, what relational algebra gives final DB query results

• Provenance
  • Every early result can be traced back to its contributing inputs

• Scalability
Thank You.

Questions??